Sentiment Analysis of Twitter Data

Semester Project Report

Kshitij Minocha, UG201310043
Rajan Vaghela, UG201310039

1. Introduction

In this project, we look at one such popular micro-blog called Twitter and build models for classifying "tweets" into positive, negative and neutral sentiment. We build models for two classification tasks: a binary task of classifying sentiment into positive and negative classes and a 3-way task of classifying sentiment into positive, negative and neutral classes. We experiment with three types of models: unigram model, a feature based model and a tree kernel based model. For the feature based model we use some of the features proposed in past literature and propose new features. For the tree kernel based model we design a new tree representation for tweets. We use a unigram model, previously shown to work well for sentiment analysis for Twitter data, as our baseline.

Our project shows that a unigram model is indeed a hard baseline achieving over 20% over the chance baseline for both classification tasks. This feature based model that uses only 100 features achieves similar accuracy as the unigram model that uses over 10,000 features. This tree kernel based model outperforms both these models by a significant margin. We also experiment with a combination of models: combining unigrams with our features and combining our features with the tree kernel. Both these combinations outperform the unigram baseline by over 4% for both classification tasks. In this project, we present extensive feature analysis of the 100 features we propose. Our experiments show that features that have to do with Twitter-specific features (emoticons, hashtags etc.) add value to the classifier but only marginally. Features that combine prior polarity of words with their parts-of-speech tags are most important for both the classification tasks.

Thus, we see that standard natural language processing tools are useful even in a genre which is quite different from the genre on which they were trained (newswire). Furthermore, we also show that the tree kernel model performs roughly as well as the best feature based models, even though it does not require detailed feature engineering.

2. Objective

We examine sentiment analysis on Twitter data. The contributions of this project are: (1) We introduce POS-specific prior polarity features. (2) We explore the use of a tree kernel to obviate the need for tedious feature engineering. The new features (in conjunction with previously proposed features) and the tree kernel perform approximately at the same level, both outperforming the state-of-the-art baseline.

3. Motivation

Micro-blogging websites have evolved to become a source of varied kind of information. This is due to nature of micro-blogs on which people post real time messages about their opinions on a variety of topics, discuss current issues, complain, and express positive sentiment for products they use in daily life. In fact, companies manufacturing such products have started to poll these micro-blogs to get a sense of general sentiment for their product.

Many times these companies study user reactions and reply to users on micro-blogs. One challenge is to build technology to detect and summarize an overall sentiment.

4. Theory

Data Description

Twitter is a social networking and micro-blogging service that allows users to post real time messages, called tweets. Tweets are short messages, restricted to 140 characters in length. Due to the nature of this micro-blogging service (quick and short messages), people use acronyms, make spelling mistakes, use emoticons and other characters that express special meanings. Following is a brief terminology associated with tweets. Emoticons: These are facial expressions pictorially represented using punctuation and letters; they express the user's mood. Target: Users of Twitter use the "@" symbol to refer to other users on the micro-blog. Referring to other users in this manner automatically alerts them. Hashtags:
Users usually use hashtags to mark topics. This is primarily done to increase the visibility of their tweets.

Resources and Pre-processing of data

In this project we introduce two new resources for pre-processing twitter data: 1) an emoticon dictionary and 2) an acronym dictionary. We prepare the emoticon dictionary by labelling 170 emoticons listed on Wikipedia with their emotional state. For example, ":)" is labelled as positive whereas ":=(" is labelled as negative. We assign each emoticon a label from the following set of labels: Extremely-positive, Extremely-negative, Positive, Negative, and Neutral. We compile an acronym dictionary from an online resource.

The dictionary has translations for 5,184 acronyms. For example, lol is translated to laughing out loud. We pre-process all the tweets as follows: a) replace all the emoticons with their sentiment polarity by looking up the emoticon dictionary, b) replace all URLs with a tag ||U||, c) replace targets (e.g. "@John") with tag ||T||, d) replace all negations (e.g. not, no, never, n't, cannot) by tag "NOT", and e) replace a sequence of repeated characters by three characters, for example, convert cooooooooool to coool. We do not replace the sequence by only two characters since we want to differentiate between the regular usage and emphasized usage of the word.

Prior polarity scoring

A number of our features are based on prior polarity of words. For obtaining the prior polarity of words, we take motivation from work by Agarwal et al. (2009). We use Dictionary of Affect in Language (DAL) (Whissel, 1989) and extend it using WordNet. This dictionary of about 8000 English language words assigns every word a pleasantness score (2R) between 1 (Negative) - 3 (Positive). We first normalize the scores by diving each score by the scale (which is equal to 3). We consider words with polarity less than 0.5 as negative, higher than 0.8 as positive and the rest as neutral. If a word is not directly found in the dictionary, we retrieve all synonyms from Wordnet. We then look for each of the synonyms in DAL. If any synonym is found in DAL, we assign the original word the same pleasantness score as its synonym. If none of the synonyms is present in DAL, the word is not associated with any prior polarity. For the given data we directly found prior polarity of 81.1% of the words. We find polarity of other 7.8% of the words by using WordNet. So we find prior polarity of about 88.9% of English language words.

5. Design

We design a tree representation of tweets to combine many categories of features in one succinct convenient representation. For calculating the similarity between two trees we use a Partial Tree (PT) kernel first proposed by Moschitti (2006). A PT kernel calculates the similarity between two trees by comparing all possible sub-trees. This tree kernel is an instance of a general class of convolution kernels. Convolution Kernels, first introduced by Haussler (1999), can be used to compare abstract objects, like strings, instead of feature vectors. This is because these kernels involve a recursive calculation over the "parts" of abstract object. This calculation is made computationally efficient by using Dynamic Programming techniques. By considering all possible combinations of fragments, tree kernels capture any possible correlation between features and categories of features.

Figure 1 shows an example of the tree structure we design. This tree is for a synthesized tweet: @Fernando this isn't a great day for playing the HARP! :). We use the following procedure to convert a tweet into a tree representation: Initialize the main tree to be "ROOT". Then tokenize each tweet and for each token: a) if the token is a target, emoticon, exclamation mark, other punctuation mark, or a negation word, add a leaf node to the "ROOT" with the corresponding tag. For example, in the tree in Figure 1 we add tag ||T|| (target) for "@Fernando", add tag "NOT" for the token "n't", add tag "EXC" for the exclamation mark at the end of the sentence and add ||P|| for the emoticon representing positive mood. b) if the token is a stop word, we simply add the subtree " (STOP ('stop-word'))" to "ROOT". For instance, we add a subtree corresponding to each of the stop words: this, is, and for. c) if the token is an English language word, we map the word to its part-of-speech tag, calculate the prior polarity of the word using the procedure described in section 5 and add the subtree (EW ('POS' 'word' 'prior polarity')) to the "ROOT". For example, we add the subtree (EW (JJ great POS)) for the word great. "EW" refers to English word. d) For any other token <token> we add subtree "(NE (<token>))" to the "ROOT". "NE" refers to non-English.

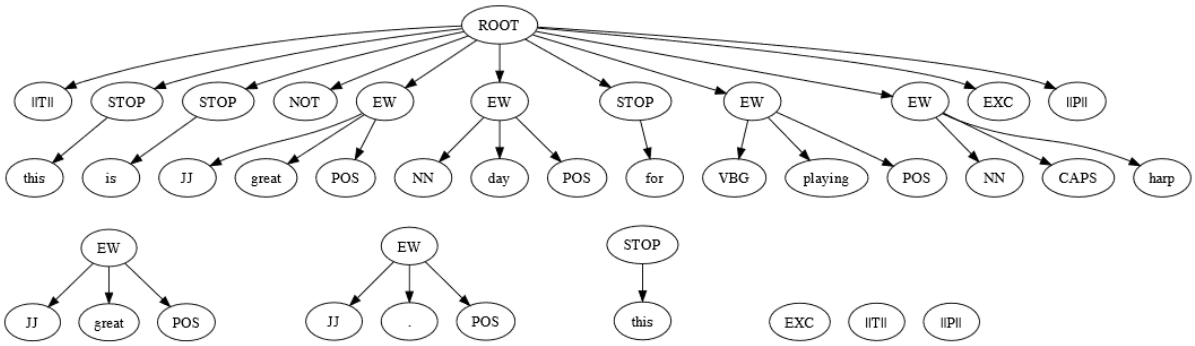


Figure 1: Tree kernel for a synthesized tweet: "@Fernando this isn't a great day for playing the HARP! :)"

6. Implementation

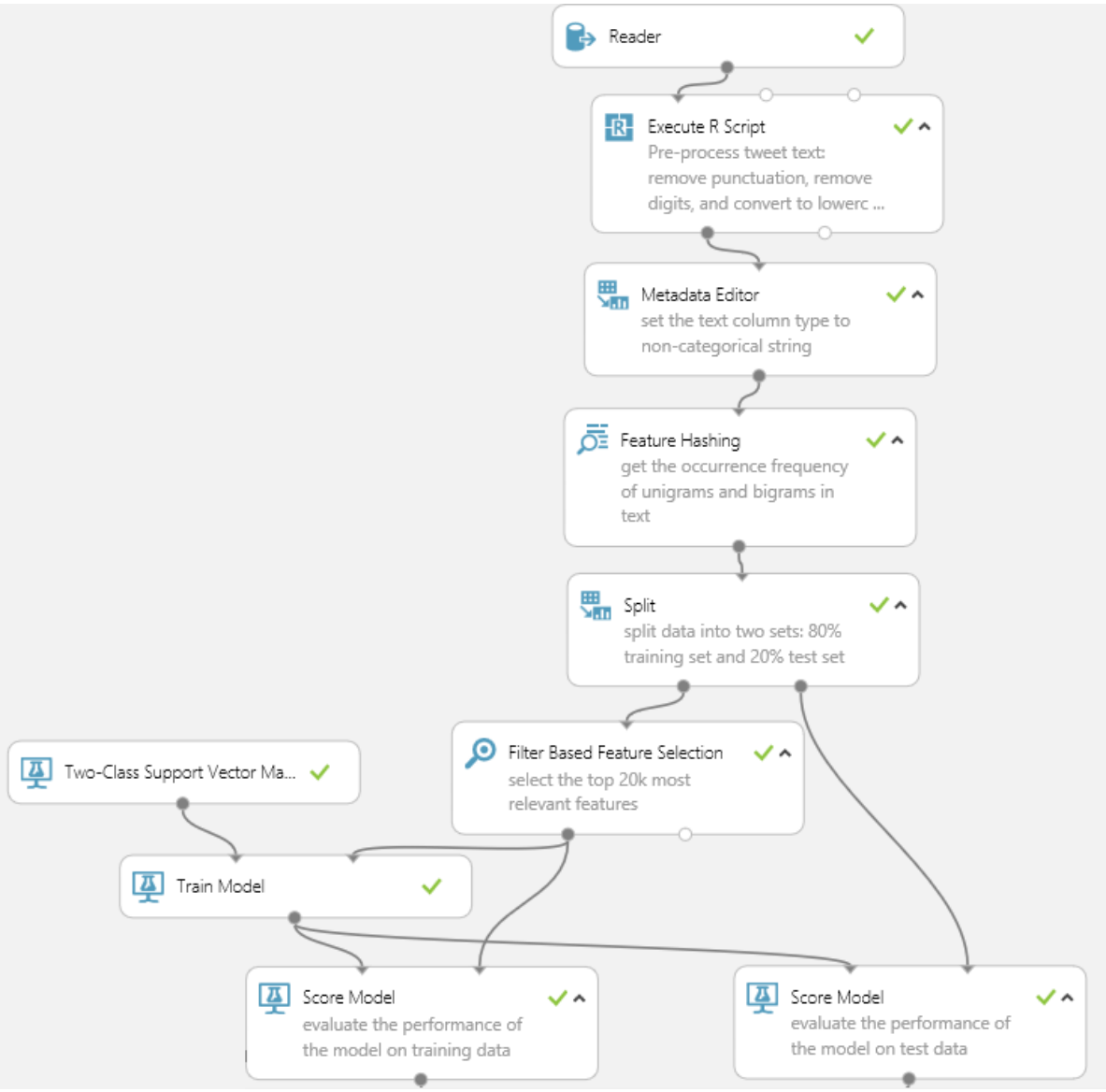
In this section, we present experiments and results for classification task:

Positive versus Negative versus Neutral.

For our project we use Support Vector Machines (SVM) and report averaged 5-fold cross-validation test results. We tune the C parameter for SVM using an embedded 5-fold cross-validation on the training data of each fold, i.e. for each fold, we first run 5-fold cross-validation only on the training data of that fold for different values of C. We pick the setting that yields the best cross-validation error and use that C for determining test error for that fold. As usual, the reported accuracies is the average over the five folds.

6.1 Analysis

We use a data-driven machine learning approach instead of a lexicon-based approach, as the latter is known to have high precision but low coverage compared to an approach that learns from a corpus of annotated tweets. The hashing features are used to train a model using the Two-Class Support Vector Machine (SVM), and the trained model is used to predict the opinion polarity of unseen tweets. The output predictions can be aggregated over all the tweets containing a certain keyword, such as brand, celebrity, product, book names, etc. in order to find out the overall sentiment around that keyword. The state diagram of analysis experiment is following.



The main steps of the experiment are:

Step 1: Get data

Step 2: Text preprocessing using R

Step 3: Feature engineering

Step 4: Split the data into train and test

Step 5: Train prediction model

Step 6: Evaluate model performance

Step 7: Publish prediction web service

6.1.1 Get Data

The dataset used in this experiment is a publicly available data set [7]. The data comprises approximately 1,600,000 automatically annotated tweets. The tweets were collected by using the Twitter Search API and keyword search. During automatic annotation, any tweet with positive emoticons, like :), were assumed to bear positive sentiment, and tweets with negative emoticons, like :(, were supposed to bear negative polarity. Tweets containing both positive and negative emoticons were removed.

For this experiment, we extracted a 10% sample of the data and shared it as a public Blob in a Windows Azure Storage account.

Each instance in the data set has 6 fields:

sentiment_label - the polarity of the tweet (0 = negative, 2 = neutral, 4 = positive)

tweet_id - the id of the tweet

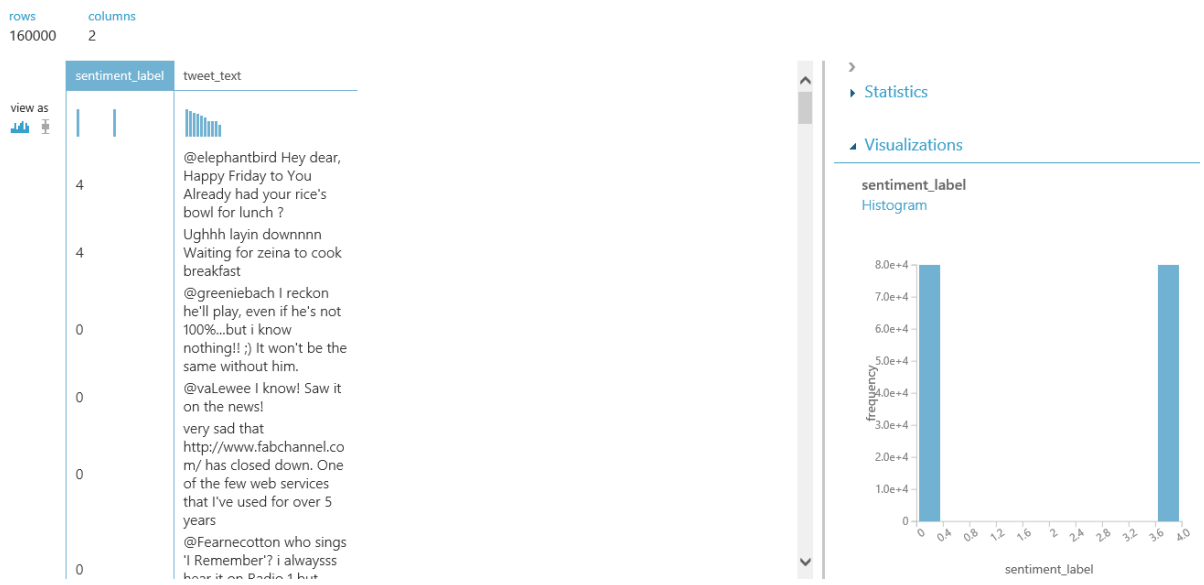
time_stamp - the date of the tweet (Sat May 16 23:58:44 UTC 2009)

target - the query (lyx). If there is no query, then this value is NO_QUERY.

user_id - the user who posted the tweet

tweet_text - the text of the tweet

We have uploaded in the experiment only the two fields that are required for training as shown below:



6.1.2 Text preprocessing using R

Unstructured text such as a tweet usually requires some pre-processing before it can be analysed. We used the following R code to remove punctuation marks, special character and digits, and then performed case normalization:

```
# Map 1-based optional input ports to variables

dataset <- maml.mapInputPort(1) # class: data.frame


# Separate the label and tweet text

sentiment_label <- dataset[[1]]

tweet_text      <- dataset[[2]]


# Replace punctuation, special characters and digits with space

tweet_text <- gsub("[^a-z]", " ", tweet_text, ignore.case = TRUE)


# Convert to lowercase

tweet_text <- sapply(tweet_text, tolower)


data.set <- as.data.frame(cbind(sentiment_label,tweet_text),

        stringsAsFactors=FALSE)


# Select data.frame to be sent to the output Dataset port

maml.mapOutputPort("data.set")
```

After the text was cleaned, we used the Metadata Editor module to change the metadata of the text column as follows.

We marked the text column as non-categorical column.

We also marked the text column as a non-feature.

The reason is that we want the learner to ignore the source text and not use it as a feature when training the model, but rather to use the extracted features that we build in the next step.

6.1.3 Feature engineering

Feature hashing

The Feature Hashing module can be used to represent variable-length text documents as equal-length numeric feature vectors. An added benefit of using feature hashing is that it reduces the dimensionality of the data, and makes lookup of feature weights faster by replacing string comparison with hash value comparison.

In this experiment, we set the number of hashing bits to 17 and the number of N-grams to 2. With these settings, the hash table can hold 2^17 or 131,072 entries in which each hashing feature will represent one or more unigram or bigram features. For many problems, this is plenty, but in some cases, more space is needed to avoid collisions.

Feature selection

The classification complexity of a linear model is linear with respect to the number of features. However, even with feature hashing, a text classification model can have too many features for a good solution. Therefore, we used the Filter Based Feature Selection module to select a compact feature subset from the exhaustive list of extracted hashing features. The aim is to reduce the computational complexity without affecting classification accuracy.

We chose the Chi-squared score function to rank the hashing features in descending order, and returned the top 20,000 most relevant features with respect to the sentiment label, out of the 2^17 extracted features.

6.1.4 Split the data into train and test

The *Split* module in Azure ML is used to split the data into train and test sets where the split is stratified. The stratification will maintain the class ratios into the two output groups. We use the first 80% of the sample tweets for training and the remaining 20% for testing the performance of the trained model.

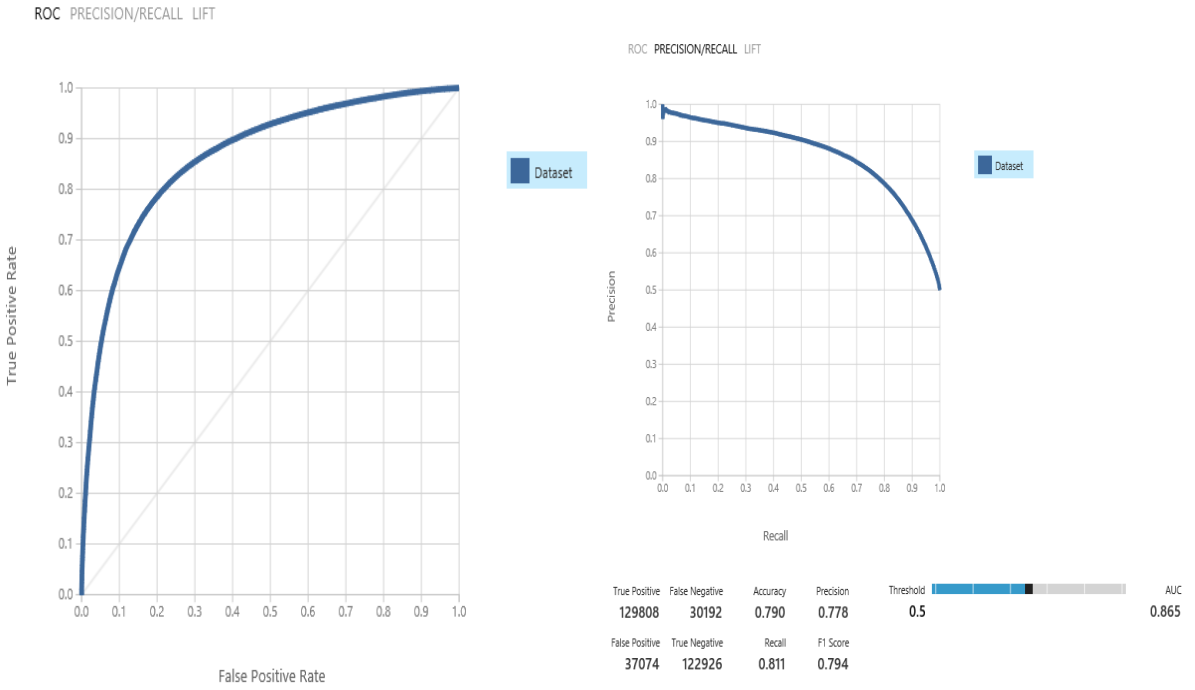
6.1.5 Train prediction model

To train the model, we connected the text features created in the previous steps (the training data) to the Train Model module. Microsoft Azure Machine Learning Studio supports a number of learning algorithms but we selected SVM.

6.1.6 Evaluate model performance

In order to evaluate the generalization ability of the trained Support Vector Machine model on unseen data, the output model and the test data set are connected to the *Score Model* module in order to score the tweets of the test set. Then connect the out predictions to the *Evaluate Model* module in order to get a number of performance evaluation metrics as shown below.

Finally, we added the Evaluate Model module, to get the evaluation metrics (ROC and precision/recall) shown in the following charts.



6.1.7 Publish Web Service

Next We created Web service of this SVM model and generated WebAPI which takes tweet_text as parameter and gives scored labels along with scored probabilities in response,scored labels can be positive, negative or neutral.

6.2 Prototype

We made prototype of this project using Asp.net MVC Website. It uses data analysis WebAPI that we made in analysis part to predict sentiment of the tweet. We also hosted this app on web using Microsoft Azure Cloud Services.

7. Conclusion

We presented results for sentiment analysis on Twitter. We use previously proposed state-of-the-art unigram model as our baseline and report an overall gain of over 4% for classification task: 3-way positive versus negative versus neutral. We presented a comprehensive set of experiments for these tasks on manually annotated data that is a random sample of stream of tweets. We investigated SVM model. For our feature-based approach, we do feature analysis which reveals that the most important features are those that combine the prior polarity of words and their parts-of-speech tags. We tentatively conclude that sentiment analysis for Twitter data is not that different from sentiment analysis for other genres.

8. Acknowledgements

We would like to acknowledge K.R. Chowdhary sir for his indispensible guidance throughout the course of our project.

9. References

1. Apoorv Agarwal, Fadi Biadsy, and Kathleen Mckeown. 2009.
   Contextual phrase-level polarity analysis using lexical affect scoring and syntactic n-grams. Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009), pages 24–32, March.

2. Luciano Barbosa and Junlan Feng. 2010. Robust sentiment detection on twitter from biased and noisy data. Proceedings of the 23rd International Conference on Computational Linguistics: Posters, pages 36–44.

3. Adam Bermingham and Alan Smeaton. 2010. Classifying sentiment in microblogs: is brevity an advantage is brevity an advantage?
4. ACM, pages 1833–1836.

5. C. Fellbaum. 1998. Wordnet, an electronic lexical database. MIT Press.

6. Michael Gamon. 2004. Sentiment classification on customer feedback data: noisy data, large feature vectors, and the role of linguistic analysis. Proceedings of the 20th international conference on Computational Linguistics.

7. Alec Go, Richa Bhayani, and Lei Huang. 2009. Twitter sentiment classification using distant supervision. Technical report, Stanford.

8. David Haussler. 1999. Convolution kernels on discrete structures. Technical report, University of California at Santa Cruz.

Resources

Code on Github: https://github.com/rajan36/Twitter-Data-Analysis
WebsieLink- http://twitteraiproj.azurewebsites.net/